ESE507 Project 1 Report

Student Jinxing Yin
ID 112263441
Student Brian Cheung
ID 112797337

Part 1: Warmup Questions

Question 1

The propagation delay is calculated through the critical path which goes from signal b into first flip flop and then the combinational logic and then a second combinational logic and lastly another flip flop. Thus, it takes a signal 4ns to go from d to q for first flip flop, 7 ns to go through two combinational logic and eventually a 3ns setup time for the last flip flop, in total the shortest clock period is 14ns. Therefore, the fastest possible clock frequency will be 1/(the clock period) which will approximately be 71.43 MHz.

There is no hold time violation. The full period of the clock signal is 14ns which means half period will be 7 ns. The hold time only requires the data signal to be stable for 1 ns, The data can change after that 1ns. By next clock cycle, the flip flop will be able to distinguish different data inputs and generate corresponding outputs. If the hold time increases to 10ns, this will require the signal to be stable longer than half period of the clock, plus there is a 3ns set up time requirement, thus, it only allows the signal to change within the 1ns time slot otherwise it will either violate the setup time or violate the hold time. This will force the input signal must behave in this way, otherwise just a bunch of violations. The most straight forward approach will be increasing the clock period to have half of the period be greater than or equal to the hold time requirement. As a result, a clock period with at least 20ns is needed.

Question 2

The problem occurs when each switching case gets executed only one output has assigned value, the other three have no assigned values which will cause inferred latches. One possible solution is to assign all four outputs before case statement with assigned values. The other way will be getting into each case statement and have all four outputs being assigned every case.

Question 3

The problem arises where output signal b being assigned by multiple always comb blocks. One solution will be “or” the assignment to signal b in the second always_comb block with the assignment to it in the first always_comb block.

Part 2 Multipile and Accumulation

4a)

The SystemVerilog testbench takes inputs from a text file to then simulates those inputs in the testbench. The text file of inputs is generated with a Python script that randomly generates in a uniform pattern. The reset input has been made to favor to not reset as frequently with previous tests demonstrating that it works correctly and needed to testthe behaviour more so when it is not being reset where inputs are being computed instead of being cleared. The Pyton script also generates an output using a system that performs similarly to the SystemVerilog MAC design to compare against the SystemVerilog testbench using the same text file randomly generated inputs. The testbench design could have been made more robust with the random inputs generated inputs being generated to favor one side or another and change after some time to more better tests system behaviour when signals such as valid_in and the enable control signals do not change for a long period of time given that the python script is mainly a uniform distribution with the randomness. Given the number of test cases being 1000 that is being generated, this is not an issue, but for fewer test cases such as 50, this can become an issue due to randomness.

```
[jyin@lab01 Part2]$ ls
adder.sv     gates.v        output.txt          runsynth.tcl   vsim.wlf
alib-52      inputData      outputData          runsynth.tcl~  work
command.log  multiplier.sv  part2_tb_simple.sv  setupdc.tcl    work_synth
control.sv   my_tb.sv       register14b.sv      system.sv
default.svf  outValues      register28b.sv      transcript
[jyin@lab01 Part2]$ diff -b outValues outputData
1d0
< x              x
1002,1003c1001
< 1       -6568647
< 0       -6568647
---
>
[jyin@lab01 Part2]$
```

Figure 1: Diff command to compare two output files from SystemVerilog and Python for Part 2.

Although there supposed to be no difference between the two files, the differences occur because the reset is synchronous so there are x at time 0 from the system verilog output. Furthermore, since the output is propagated two clock cycles with respect to the input, the system verilog waited until the last two finished. On the other hand, the Python design had paired relation between input and output, thus it did not produce the output of the last two set of input. Given the number of test cases used, the difference between using 998 sets of inputs and verifiable outputs and 1000 sets is negligible so this solution was accepted. Further test cases can

be added for a more exhaustive test but was deemed unnecessary with the 1000 generated inputs testing all necessary test cases.

4b)

To cause the accumulator to overflow a separate text file was created with the reset signal forced to be clear for all tests cases. Doing this will eventually accumulate to sums that will overflow or underflow the maximum and minimum limits that the 28-bit accumulator can handle. To detect when the system would overflow in the SystemVerilog you can detect if the previous accumulation that has a set bit on the 26th bit becomes cleared after the summation meaning that the sum has been truncated or became negative due to overflow. For underflow you would check if the 27th bit that was set becomes cleared after the summation meaning that the sum has been truncated due to underflow.

4c)

Frequency (MHz)	Area (μm^2)	Power (μW)
869.57	1498.91	731.06
840.34	1512.74	707.41
833.33 (best)	1533.76	711.62
800.00	1489.07	667.00

Table 1.1 Clcok Frequency the student selected and corresponding area and power synthesis report

There were several different clock frequencies chosen to be synthesized. The critical path varied a little bit for the different frequencies that were selected. Generally, the critical path starts from the D flip flop of either register A or B and then goes to a buffer. After that, there is a set of combinational logic circuit elements which comprises the multiplier and adder logic (e.g. XNOR2, ORANDINVERT2, NOR2, NAND2, ANDORINVERT2, Full Adder), eventually the signal passes through the output register. The first frequency chosen is 800 MHz (this frequency was chosen at random) and it turned out that the timing requirement was met with some extra time. Thus the next frequency chosen is 833.33 MHz and turned out to be the best frequency. There is still a 0.01 time extra in slack so a higher frequency was chosen, 869.57MHz but failed the timing requirement. The frequency was lowered to 840.34 MHz which also failed the timing requirement, since it was only 0.01ns different from 833.33MHZ for the clock period, the conclusion is that the 833.33 MHz was the highest frequency which could pass the timing requirment.

4d)

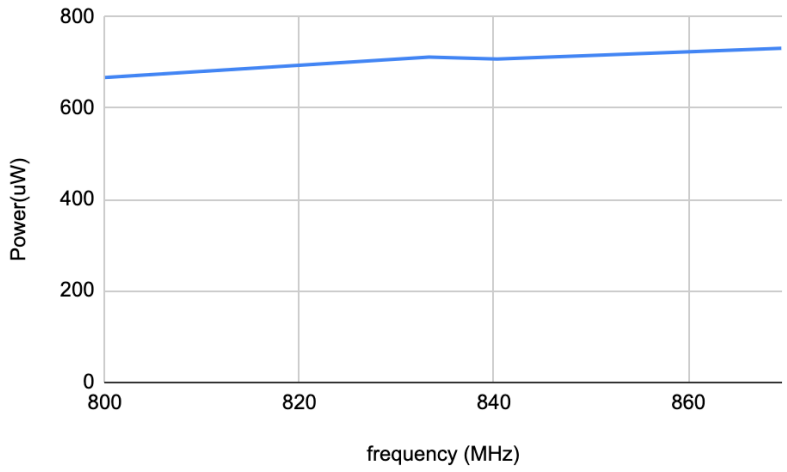

Figure 2.1: Frequency and Power Graph

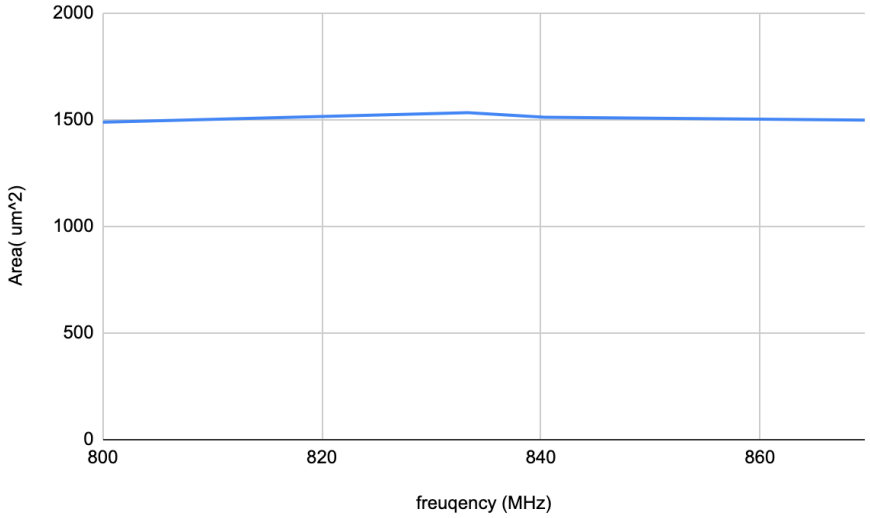

Figure 2.2: Frequency and Area Graph

The above two graphs show the relation between the frequency with area and power. From both graph, one can see that, as the frequency increases both area and power increases. Although for the relation between frequency and area, it's not very clear, sometimes increasing the frequency also lower the area a little bit. From my perspective for the synthesis report of the same circuit with different frequency, the area does not change a lot as there is not direct relation between them. Furthermore, the area estimation does not taking account the wiring as a normal

circuit, thus the relation between frequency and area is debatable. On the other hand, the power increases as the frequency increases, they have direct relation (proportional) to each other.

4e)

$$E = P * t = 711.62 * 52 * 1.2 = 44405.09\ \mu W*ns$$

4f)

If the student change the frequency, the energy will change. The reason is that although when calculating the energy we multiplier the power with time, power is proportional to frequency and frequency is reverse of the time, thus, they get cancelled. Therefore, the only factors affect the energy will be the capacity and voltage, if we change the frequency the capacity will slightly changes, thus the energy will also change.

4g)

It's necessary for the students to include the reset signal for every register. There were in total five registers were used in this part of the design. If any of them did not have the reset signal, suppose someone entered the design and wanted to reset the system. With the absence of all registers being reseted, the people would find out that the system produced some incorrect output values.

Part 3: Saturating Arithmetic

```
# run -all
# ** Note: $finish    : my_tb.sv(41)
#    Time: 10025 ns  Iteration: 1  Instance: /my_testbench
# End time: 16:10:13 on Oct 02,2022, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
[jyin@lab01 Part3]$ ls
Pout     command.log  gates.v    my_tb.sv   output.txt    runsynth.tcl~  system.sv   work
alib-52  default.svf  inputData  outValues  runsynth.tcl  setupdc.tcl    transcript  work_synth
[jyin@lab01 Part3]$ diff -b outValues Pout
1d0
< x              x
1002,1003c1001
< 0              0
< 1      -22379900
---
>
[jyin@lab01 Part3]$
```

Figure 3: Diff command to compare two output files for Part 3. The Python code does not simulate the unknown state at the start and the last 2 cycles after the last input hence the

difference that appears in Figure 3. The SystemVerilog and Python code otherwise do match and can be verified to have the correct outputs.

4a)

The overflow/underflow detection can be classified into two situations either both inputs were negative and the output was positive or both input were positive and the output was negative. Therefore, a checking condition in the adder to account for the overflow/underflow. If both inputs into the adder are positive and the result was negative, the result would be overwritten to hold the maximum positive value which was 28'h7fffff = 134,217,727. On the other hand, if both input are negative and the output was positive, then the output would be saturated into 28'h8000000 = –134,217,728.

4b)

Similar to Part 2, the reset can be forced to be clear for all tests cases, at some point with sufficient test cases (in our tests, 1000 tests cases were used and was sufficient) it will hit the upper or lower limit that will saturate to the highest or lowest possible value. In Python the random input generation remains the same, what has been changed is when the value is being accumulated, instead of truncating, the value is checked to see if it goes beyond the maximum or minimum possible value. The SystemVerilog testbench will use these random inputs from the text file to get an output to verify against the Python output for correctness of saturation behaviour.

4c)

The maximum frequency the student was able to reach was 645.16 MHz, and the power was 605.8 μW and the area was 1731.9 μm^2. The critical path followed along the same route except there were more circuit element added to implement the checking process of the saturation. Generally, the critical path started from register a or b, then went through the multiplier. After that, the signaled went over the adder and eventually got into the output register. That was where critical path located. In comparison to part two, as more circuit elements were added, longer clock period was required, thus lower clock frequency it could achieve. It was observed that the critical path got longer. Since there were more circuit element, the area would also increase. On the other hand, the power is directly proportional to frequency, if frequency got dropped down, so did power.

4d)

$E = P * t = 605.8 * 52 * 1.55 = 48827.48\ \mu W*ns$

$48827.48\ \mu W*ns > 44405.09\ \mu W*ns$

The design in Part 3 requires more energy in comparison to the Part 2 design does.

Part 4: Pipelining

Part 4.1

Adding an extra register will make the valid out signal and accumulation result for a specific time getting one more clock cycle delayed. The testnech also need to change to adapt this behavior. The frequency was 862.1 MHz, the correspnding area was 1794.4 and the power was 918.3 μW. The critical path went from the output coming out of either register a or b, and then passing through the multiplier circuit and lastly entered the pipeline register.

```
# Errors: 0, Warnings: 1
[jyin@lab01 Part4b]$ ls
Pout     command.log  gates.v    my_tb.sv   output.txt     runsynth.tcl   setupdc.tcl  work
alib-52  default.svf  inputData  outValues  part4b_mac.sv  runsynth.tcl~  transcript   work_sy
[jyin@lab01 Part4b]$ nano Pout2
[jyin@lab01 Part4b]$ diff -b outValues Pout2
1d0
< x             x
1002,1003d1000
< 0     -100067270
< 1     -133940165
[jyin@lab01 Part4b]$
```

Figure 4.1: Diff command to compare two output files for Part 4.1.

Part 4.2

```
Pout   Pout4  alib-52      gates.v    outValues     runsynth.tcl   transcript  work_synth
Pout2  Pout5  command.log  inputData  output.txt    runsynth.tcl~  vsim.wlf
Pout3  Pout6  default.svf  my_tb.sv   part4_mac.sv  setupdc.tcl    work
[jyin@lab01 Part4a]$ nano Pout7
[jyin@lab01 Part4a]$ diff -b outValues Pout7
1d0
< x             x
1002,1003d1000
< 0             0
< 0             0
```

Figure 4.2: Diff command to compare two output files for Part 4.2.

Question 4.3)

Number of stage	Frequency (MHz)	Area(μm^2)	Power((μW))
4a, just one pipeline register	862.07	1794.44	918.3

Pipeline plus 2stage multiplier	**892.86**	**1950.05**	**1182.2**
Pipeline plus 3stage multiplier	**1123.60**	**2230.94**	**1944.5**
Pipeline plus 4 stage multiplier	**1149.43**	**2363.41**	**2118.4**
Pipeline plus 5 stage multiplier	**1123.60**	**2436.83**	**2281.8**
Pipeline plus 6 stage multiplier	**1176.47**	**2565.57**	**2608.5**

Table 4.1 Different stages multiplier the student selected and corresponding frequency, area and power synthesis report

In part4, as the student pipelined the stage in between the adder and multiplier, the student had to also take care of the control system to generate the proper enable signals and valid out signal. The methodology the student followed was whenever there was an extra pipeline register, the valid out would be delayed by one more clock cycle, and the accumulation result would be also delayed by one more clock cycle. The enable register to the output register would also need to adapt the changes. The critical path for 4a was the looping back path from the output register going back to itself. The 2 stage one had a critical path from the pipeline register to the output register. The 3 stage, 4 stage both have the critical path inside the pipelined multiplier.
For 5 stage, the critical path was looping through the output register and the adder. For 6 stage, the critical path was coming from the pipeline register to the output register.

For just 1 pipeline register E = P * t = 918.3 * 52 * 1.16 = 55391.86 μ*W*ns
For pipeline register plus 2 stage multiplier E = P * t = 1182.2 * 52 * 1.12 = 68851.33 μ*W*ns
For pipeline register plus 3 stage multiplier E = P * t = 1944.5 * 52 * 0.89 = 89991.46 μ*W*ns
For pipeline register plus 4 stage multiplier E = P * t = 2118.4 * 52 * 0.87 = 100360.42 μ*W*ns
For pipeline register plus 5 stage multiplier E = P * t = 2281.8 * 52 * 0.89 = 105601.7 μ*W*ns
For pipeline register plus 6 stage multiplier E = P * t = 2608.5 * 52 * 0.85 = 115295.7 μ*W*ns

Question 4.4)

There is no unique answer to this question. If the student prefers to have a MAC system which can have more operation rate then, the student will choose to pipeline the system to have a shortest possible clock period which is the MAC with 6 stage multiplier. On the other hand, if the student want to have a MAC system which has least latency, then the student will choose to not pipeline the circuit to have multiple stages which would be part 3 design. Also, the power and area will also be important factors to consider, however , they are related to the frequency.

Question 4.5)

It's possible but not quite feasible to adder pipeline into the adder. It's part of the looping back functionality, if it gets pipilined the behavior of that looping back will be very hard to distinguish or event not work properly.

Work Distribution)

Student Jinxing Yin worked on the SystemVerilog code and Report Formation
Student Brian Cheung worked on the Python testbench and Report Formation